Getting started with flutter.

void main - The main code begins here.

runApp - It is used to tell that the app should run from here.

materialApp - It is used to make a material design app (It has many prop.)

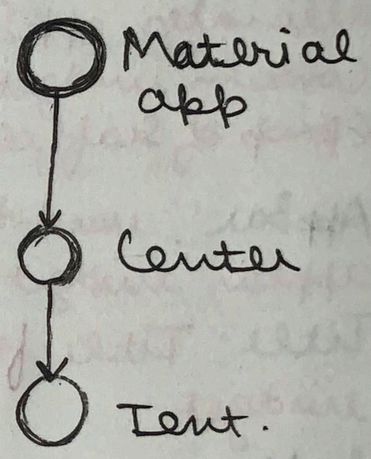
home - Home is where everything starts

Center - Center widget brings the child ele. to center.

Text - It is used to add text to app.

```
import 'package:flutter/material.dart';

void main () {
    runApp(
      MaterialApp(
        home: Center(
          child : Text ('Hello world'),
          ),
        ),
      );
    }
```



** Everything in flutter is a widget

→ Here in this code a text hello world is displayed on the main screen in the center of screen.

Scaffold Class

Scaffold - It is a Class used to arrange diff. ele. in the app (have many properties.)

backgroundColor - used to change backgrond colour of present widget.

AppBar - It is used to tell Scaffold that the place where it allocates appBar will Contain widget it hold. (prop of Scaffold)

AppBar: used to specify appbar widget

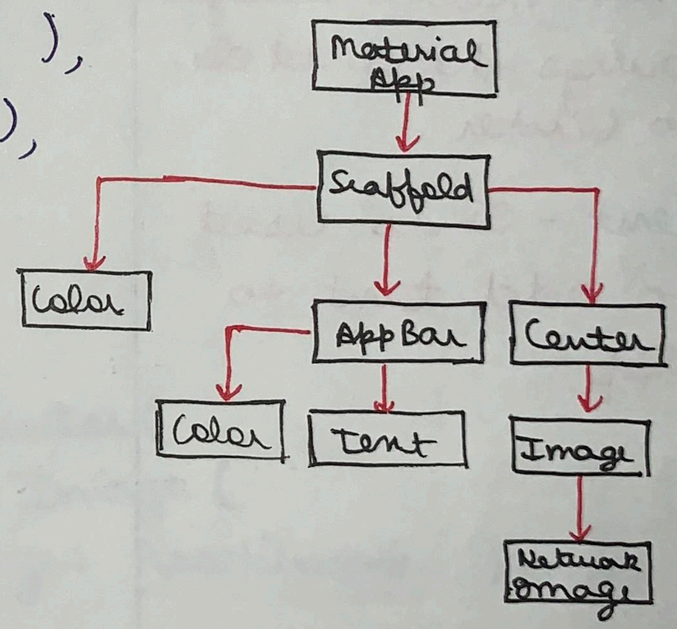
Title: Title for present widget

Body - The area below appBar.

Image - will make an image widget with no image.

NetworkImage/Asset Image - used to tell that the image location is from internet or from project folder.

```
import 'Package: flutter /material.dart';
void main () {
  runApp(
    MaterialApp(
      home: Scaffold(
        backgroundColor: Colors. blue,
        appBar: AppBar (
          title: Text ('I Am Rich'),
          backgroundColor: Colors.Blue,
        ),
        body: Center (
          Child: Image (
            image:
              NetworkImage('url'),
          ),
        ),
      ),
    ),
  );
}
```



- Scaffold is used to arrange the elements of app properly. It contains properties that specify the location of a particular widget.

** Draw.io (to create flowcharts)

assets - This is used to specify that what resources are being used in the app from app files.

AssetImage - AssetImage is used to add image from the app resources file rather than from internet. (Network image).

File location:

appname → pubspec. yaml.

** To add images from your app project folder you need to specify the folder path in pubspec. yaml file.

```
assets:
  - images/
```

** The child should have 2 spaces from the start of parent.

** after specifing the image folder path in yaml. add some text to main.

```
void main() {
  runApp(
    ),
    Body: Center(
      Child: Image(
        image: AssetImage('path'),
      ),
    ),
  ),
```

** Adding path of assets to yaml file so that we can use the files in our application.

Hard refresh / Reload.

MyApp() - It is the function that is executed by runApp.

StatelessWidget - This widget is used to make the app display change in live.

Build - It is method that will be called everytime we change the code.

return - The code placed in after return will be returned By the function to where it is being called.

```
import 'package: flutter/material.dart';

void main() {
    runApp(MyApp());
}

Class MyApp extends StatelessWidget {
    @override
    Widget build (BuildContext context){
      return MaterialApp (
        home: Scaffold (
            backgroundColor: Colors.white,
            Body: Container();
        ), // Scaffold
      ); // MaterialApp
    }
}
```

** type Stless for boiler plate for Stateless widget

** The code will run from void main the the runApp will call MyApp class and it inherit StatelessWidget which is used for live preview. Then It starts and every time we change anything build function is called. Then it will return the code to the place where function is being called.

Container in flutter.

Safearea - The safearea is a widget which makes the content visible in main visible area (no notch area).

Container - container acts as a div in web. It can only contain one child class.

height & width - It is used to set the height and width of container.

margin & padding - They are used to give margin and padding to the container widget

EdgeInsets - This class is used to provide specify where margin and padding is used (prop.))

only, all etc - EdgeInsets class provide diff objects to apply margin and padding to specific places.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.teal,
        body: SafeArea(
          child: Container(
            height: 100.0,
            width: 100.0,
            margin: EdgeInsets.only
                (left: 30.0),
            Padding: EdgeInsets.all
                (20.0),
            color: Colors.white,
            Child: Text('Hello'),
          ),
        ),
      ),
    );
  }
}
```

→ This code will create an app with background colour teal and will contain a div below notch (if any) due to Safearea widget. and will have margin and padding.

** "all" object of class EdgeInsets will apply padding to all edges.

** "only" object of class EdgeInsets will apply margin on left.

Column and Row widget.

Column - column will contain container and children in vertical format while row in horizontal format.

CrossAxisAlignment: It is used to set layout of container ie children in diff manner and we can use other prop. too instead of this

Children - Column contains many things so instead of child it uses children as a widget.

<widget>: It is used to tell that it contains many widgets in it

SizedBox - It is used to give space b/w diff containers.

```
home: Scaffold(
  backgroundColor: Colors.teal,
  body: SafeArea(
    child: Column(
      CrossAxisAlignment: CrossAxisAlignment.stretch,
      Children: <Widget>[
        Container(...),
        SizedBox(
          heigh: 20.0,
        ),
        Container(...),
        Container(...),
      ], // Container
    ], // <widget> []
  ), // Column
), // SafeArea
), // Scaffold.
```

** Flutter layout cheat sheet

more layout practice

** This will contain 3 div stretched fully and covering whole screen. SizedBox will provide a gap of 20 point B/w 1st and 2nd Container.

Tapping into widget properties.

Circle Avatar - This widget is used to give a profile picture like feel on layout.

radius - It is used to give radius of Circular avatar

Style - Style widget is used give style

TextStyle: This widget is used to provide Style for the text It contains many prop.

```
home : Scaffold (
    backgroundColor : Colors . teal,
    body : SafeArea (
        child : Column (
            Children : < widget > [
                CircleAvatar (
                    radius : 50.0,
                    backgroundImage : AssetImage ('url'),
                ),
                Text (
                    'Sanchit',
                    Style : TextStyle (
                        fontSize : 40.0,
                        Color : Colors . white,
                        fontWeight : FontWeight . bold,
                    ),
                ),
            ],
        ),
    ),
);
```

** ctrl + q to get prop of the widget

→ This code will display a profile picture picked from Assets and will show a text written below the profile picture.

Custom fonts.

Font family - font family is the style of fonts (prop of Style).

** first we have to download the font from flutter fonts then will make folder and shift the downloaded file to a new folder and will add the folder path to .Yaml file.

```
body: SafeArea(
    child: column(
    Children: <Widget>[
        CircleAvatar(
            radius: 50.0,
            backgroundImage: AssetImage
                                ('Path'),
        ),
        Text(
            'Sanchit',
            Style: TextStyle(
                fontFamily: 'Pacifico',
                fontSize: 40.0,
                color: Colors.white,
                fontWeight: FontWeight.bold,
            ),
        ),
        Text(
            'Flutter',
            Style: TextStyle(
                fontFamily: '';
            ),
        ),
    )
```

→ Additional code for fontfamily will change the font style to the specified font style.

Adding Material icons.

Icon - This widget is used to create icon. (icons are drawn on app rather than only display like images)

Icons - This class is used to create diff icons already provided by header (import 'package: flutter/material.dart';)

Icon widget contain diff prop too.

** use material pallate for flutter to make combinations and icons

```
home: Scaffold(
  BackgroundColor: Colors.teal,
  body: SafeArea(
    child: column(
      Children: <Widget>[
        CircleAvatar(...),
        Text(...),
        Text(...),
        Container(...),
        Container(...),
        Container(
          Color: Colors.white,
          Margin: EdgeInserts.Symmetric(
            horizontal: 30.0,
          ),
          child: Row(
            children: <widget>[
              Icon(
                Icons.mail,
                Color: Colors.teal[800],
                Size: 25.0,
              ),
              Text(
                '———',
                Style: TextStyle(

                ),
            ),
```

** On this code we have Column in containers and Containers in column. with combinations of them we can make intresting and good layouts.

** Column doesn't have background colour option but container has.

Cards, padding widget, divider.

Card - Card is a pre defined widget that contains icons and texts to display. It is used to keep the material designed app material only.

ListTitle: This widget is used set elements into the ListTitle widget.

leading - with this you can set image or icon at start of list.

Padding - Padding widget provide a layer of padding around its child.

divider - divider acts as an horizontal rule.

```
...
Card(
  margin: EdgeInsets.symmetric(vertical
                  :10.0, horizontal: 25.0),
  Child: ListTitle(
    leading: Icon(
      Icons.phone,
      Color: Colors.teal,
    ),
    title: Text(
      'Hello',
      Style: TextStyle(
        Color: Colors.teal.shade300,
        Fontfamily: 'Source Sans Pro',
        fontSize: 20.0,
      ),
    ),
  ),
Card(
  Child: Padding(
      padding: EdgeInsets.all(25.0),
      Child: (Row...),
    ),
  ),
SizedBox( child: Divider( color: Colors.tealShade
                                  100,)
)
```

** In this code we explored the combination of rows, containers, columns and used the simple way to add Cards.

Using the expanded widget

expanded: This widget is used to place content inside it to screen so that the content be aligned properly if device is changed.

flex: flex is used to make the content bigger than the other expanded class. (default flex is 1)

** if expanded class has flex 2 it will be greater than the one with flex smaller than 2.

• Image.asset: This is a short line for Image widget.

```
home: Scaffold(
    backgroundColor: Colors.red,
    appBar: AppBar(
      title: Text('Dice'),
      backgroundColors: Colors.red,
    ),
    body: DicePage(),
  ),
),
);
}

Class DicePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: <Widget>[
        expanded(
          child: Image.asset('___'),
        ),
        expanded(
          child: Image.asset('___'),
          flex:2,
        ),
      ],
    );
  }
}
```

** In this code an appBar will be displayed and in the body section there will be a row with 2 images and due to expanded widget they will take up maximum horizontal space of screen and due to flex 2 in 2nd image it will be greater than the 1st one.

Adding flutter flat button.

Flatbutton: This widget is wrapped around other widget to make them a flat Button (It has default padding).

Onpressed: This is the property of flat button which is mendory to mention.

(){} - This is void Callback funtion. Anonymus funtion is used to define what will the button do on pressed.

print: print will print the output on terminal.

```
return Center(
  Child: Row(
    Children: <widget>[
      Expanded(
        Child : Flatbutton(
          onPressed: () {
            print('left button');
          };
          Child : Image.asset('-'),
        ),
      ),
      Expanded(
        Child : FlatButton(
          on Pressed: () {
            print('right button');
          };
          Child: Image.asset('-'),
        ),
      ),
```

** In this code we will wrap the image in flatbutton widget to make it funtion like a button and on pressed helps to identify the press and will perform that particular funtion.

Making the Dice image change Reactively

Var - This is used to declare a variable to further use in program.

$ - It is used to change the value at the place where it is written and the var name specified after it value is placed in place of $.

```
Class DicePage extends StatelessWidget {
    @ Override
    Widget build (BuildContext context) {
      Var leftDiceNumber = 5;

      return Center(
        child : Row(
          children: <Widget> [
            onpressed: () {
              print('———');
            },
            child: Image.asset ('
            images/dice$ leftDicenumber .png'),
          ),
        ),
```

** In this code there is a variable with value 5 and in image path where $ is written and after it var name is written which contains number will replace the $ and will write 5 instead after compiling. So, dice5.png will be displayed

Stateful widget.

StatefulWidget -
This widget is used to make a ~~dyna~~ dynamic app. where state of widgets change on user use.

Setstate () - This widget is used to call build function to check the changes in widget, variables or anyother thing and apply to it

* Anything inside it will get change inside it and this will call build function and the place where that name is specified will be updated.

```
Class DicePage extends StatefulWidget {
    @ override
    - DicePageStage CreateState() => _DicePage
                                     State();
  }

Class - DicePageState extends State<DicePage> {
    ~~int leftDiv~~
    int leftDiceNumber = 5;
  @ override
  Widget build (BuildContext Context) {
    return Center (
      child : Row (
        Children: <Widget> [
          Expanded (
            child : FlatButton (
              onPressed : () {
                SetState (() {
                  leftDiceNumber = 2;
                  print ('dicenumber = $lef');
                });
              },
              Child: Image.asset ('images/
                dice$leftDiceNumber.Png'),
            ),
          ),
```

→ In this code we used Statful widget because it will change the state of diff widgets.
So in this when user click left dice image Setstate function will be called and leftDiceNumber will be changed to 2 and setstate will call build where build will change the variable number to every place where it is used.

Using flutter packages and functions

import 'dart: math' - This will import math package to the app so that we can use its function that are pre build.

Random - Random is a predefined class in dart: math which produces random outputs.

nextInt: this is a property or can be said object that specify the maximum number at which random should generate the random numbers from 0.

** Variables are declared here and all the functions are defined here.

** In this function it defined to change the die image when called by calling setstate. So that there is no extra code to do the something.

```
import 'dart: math';
Void main (){
    return runApp(
        MaterialApp(
            home: Scaffold(
                backgroundColor: Colors. teal,
                appBar: AppBar(
                    title: Text('Die Roller')
                    backgroundColor: Colors. teal,
                ),
                body: DiePage(),
            ),
        ),
    );
}

Class DiePage ————————————
  @override
  ———————————— => _DiePageState();
}

Class _DiePageState extends State<diePage>{
  int leftDie number = 4;
  int rightDieNumber = 5;
void changeDieFace() {
    SetState(() {
        rightDieNumber = Random(). nextInt(6) + 1;
        leftDieNumber = Random(). nextInt(6) +1;
  });
}
@override ——
    return center(...),),);
        (...)
        FlatButton(
            onpressed:() {
                ChangeDieFace();
            }.
```

using function and packages. Open source.

audio players/ — :
This is an open source package. Can be added to package of Application

Steps:
- www.pub.dart.lang
- ↓
- Choose package
- ↓
- Install option
- ↓
- add dependencies to project in pubspec. Yaml file
- ↓
- Import file to your main from readme option on package site.

final - It is used to make player for audio.
expanded return type - It will return layout to ~~from~~ where the call is made

** download Open source packages from pub.dart.lang

```
import 'Package: audioplayers/audio_cache.dart';
void main() => runApp(XylophoneApp());

Class XylophoneApp extends StatelessWidget{
  void playsound (int number) {
    final player = AudioCache();
    player.play ('note$number.wav');
  }

  Expanded buildkey ({Color color, int Number})
  {
    return Expanded (
      Child: FlatButton (
        Color: color,
        on pressed: () {
          playSound (Number);
        },
      ),
    );
  }

@ Override
Widget build (——) {
...
children: <Widget>[
buildkey(color: Colors.red, Number: 1),
],
```

function Call.

→ In this code we imported Open source package that helps in playing music from our flutter project. It contains final keyword that creates a player for music and player.play will play the sound mentioned. from main function buildkey function will be called that will return layout to the place from where it was called. buildkey will get 2 parameters that will be used to give layout and function to diff buttons. from on pressed when button is clicked play sound function

Lists data type.

List<widget> - This will create a list with a type of widget. All the children that it will contain ~~with~~ should only be type of widget.

** we can replace widget with icons, string etc.

~~Shop~~ Scorekeeper.add - This will append a child in scorekeeper list.

Row (children: Scorekeeper). This will be calling a list named scorekeeper rather than creating list here only.

```
Class _QuizPageState extends State<quizpage>{
  List<widget> scorekeeper = [ ]
    Icon(
      Icons.check,
      color: Colors.green,
    ),
    Icon(
      Icons.close,
      color: colors.red,
    ),
  ];
  @override
    return column(
      children: <Widget>[
        Expanded(...),
        Expanded(
          Child: FlatButton(
            textcolor: Colors.white,
            color: Colors.green,
            child: text(
              'True',
            ),
          ),
          onpressed: () { SetState (() {
            Scorekeeper.add(
              Icon(Icons.check, color: - ,)
            );
          });
        },
        Row (children: Scorekeeper,),
```

So, In this code we have created a list of widget before using it rather than making inside our main code so that to easily handle the operations. And by clicking on true button it will be appending a check icon to list that will show in row below.

String data type.

<String> - It is used to create list of strings that has index for every line.

** this will create a funtion that will call build after increment. So that all the ansnumbers variable get incremented

question [quesNumber]. This will show out put of text that has index quesNumber in it

```
(...)
List<String> questions = [
  'You can lead a cow down stairs',
  'A Slug\'s blood is green',
],

int quesNumber =0;
void countClicks() {
  SetState (() {
    quesNumber ++;
  });
}

Expanded (.
  Child : Text(
    questions [quesNumber],
    textAlign : TextAlign . Center,
  ),
),

Expanded (
  Child : ElevatedButton (
    Child : Text(
      'True ,),
    onpressed:() { countClicks() {
      ~~[illegible]~~,
      SetState (() {
        Scorekeeper . add(
          Icon(
            Icons . check,
            Color : colors . green,), ), }.
```

→ In this function when elevated button is clicked it will call count clicks that will increment ques number by 1 and will call build function then will be calling setstate and will add Icon on row at bottom of the app.

Checking of the answers

** In this code. we will create a list of boolean that will contain answers for quiz (previous page). we have to place true and false in exact same place of the question of which the answer is.

** when button is pressed it will do specific tasks.

If true button is pressed then it will take the value of first answer from answers list and will assign it to a var. then it will check the answer and if is ~~true will~~ right then a tick will appear in below row. and vice versa.

```
List<bool> answers = [False, true];

@override
Widget build (BuildContext context){
  Expanded (
    Child : ElatButton (
      Child : Text (
        'True',
      ),
    ),

    onpressed : (){
      bool correctAnswer = answers [questionNum];
      if (correctAnswer == true) {
        SetState(() {
          Scorekeeper.add (
            Icon(
              Icons . check,
            ),
          );
        });
      } else {
        SetState(() {
          Scorekeeper.add (
            Icon (
              Icons. close,
            ),
          );
        });
      }
      SetState (() { questNumber ++ ;
      });
    },
  ),
```

Classes and objects use

This is a new class made in new ~~Dart~~ file. Here we will define functions to call from another file by importing this file.

→ This will call the constructor of Question class and will assign these values to functions.

→ Here correctAnswer will be getting answer of diff questions by calling list and in that list calling constructor of questionNumber

```
# file - question.dart
class Question {
  String questionText;
  bool questionAnswer;

  Question ({String q, bool a}){
    questionText = q;
    question Answer = a;
  }
}
```

```
# file. main.dart
List<Question> questionBank = [
  Question (q: 'you are good', a: false),
  Question (q: 'Hello guys', a: True),
];

(...)
compressed : (){
  bool correctAnswer = questionBank [questionNumber]
                                    .questionAnswer;
(...)
Child: Text (
    questionBank [questionNumber].questionText.
```

→ In this code we will replace 2 diff lists into 1 class to make our code look clean. will be creating other file and will importing that file to our main.dart to use its function.

Setting up themes.

Theme - theme is used to set up theme for the app.

ThemeData.dark() - this will set up dark theme for the app.

- **copywith** - this will change the following things contained in it with dark theme.

primary color - this will change color of app bar.

ScaffoldBackgroundColor - This will change background color of theme of body

Theme - This will be used to set up particular theme where applied

data: ThemeData - this will contains the contents to be changed from theme applied above.

```
Void main () => runApp(BMICalc());

Class BMICalc extends StatelessWidget {
  @override
  Widget build (BuildContext context){
    return MaterialApp(
      theme: ThemeData.dark().copywith(
        primaryColor: Color (0xFF__),
        ScaffoldBackgroundColor: color(__),
      ),
      home: InputPage(),
    );
  }
}

Class InputPage extends StatefulWidget {
    return Scaffold(
      floatingActionButton: Theme(
        data: ThemeData(accentColor: Colors.deepPurple),
        child: FloatingActionButton(
          child Icon (Icons.add),
        ),
      ),
    );
}
```

→ So this will change the default app theme to dark with some customization. All the properties will be applied to the whole app. But due to specific applying theme on floating button it will change the theme of button

```
Class Card extends StatelessWidget
{ Card({@required this.colour});
final Color colour;
@override
Widget build(BuildContext
                 context){
  return Container(
    margin: EdgeInsets.all(15.0),
    decoration: BoxDecoration(
      color: colour,
      borderRadius: BorderRadius.
        circular(10.0),
      ),
    );
  }
}
```

→ So, in this code because of repetation of same code a Class named Card is made seperately to use it where needed.

```
Class _InputPageState extends State<InputPage>{
  @override
  Widget build(BuildContext context){
    return Scaffold(
      body: Column(
        children: <Widget>[
          Expanded(
            Child: Row(
              children: <Widget>[
                Expanded(
                  child: Card(
                    Colour: color(0xFF-),
                  ),
                ),
                Expanded(
                  child: Card(
                    Colour: color(0xF-),
                  ),
                ),
              ],
            ),
          ),
          Expanded(...),
        ],
      ),
    );
  }
}
```

this.colour - This will set colour of card by passing value to constructor and final keyword will make the colour immutable after setting colour once.

enum [dart]

enum Gender - This is is a datatype which is used to store values and members in it.
(This will create a list type name Gender)

Gender - This will become a datatype to be used inplace of int, string etc.

Gesture Detector - this is used to set functions to widgets. eg ontap, onslide etc.

```
import ...

enum Gender {
  male,
  female,
}
Class InputPage extends State<InputPage> {
  void updateColour(Gender selectedGender) {
    if (selectedGender == Gender.male) {
      ...
    }
  @override
  widget build(BuildContext context) { (...
  Expanded(
    child: GestureDetector(
      onTap: () {
        Setstate(() {
          updateColour(Gender.male);
        });
      },
      child: card(
        Cardchild: CardContent(
          icon: FontAwesomeIcons.mars,
          label: 'Male',
        ),
      ),
    ),
  ),
```

→ enum here will create a list that will contain data and can be used to extract and add data to it. Can be used to send and get parameters et.

Navigation from one Screen to next.

- **Initial route** - This will specify the page or screen number to be displayed on home. Both home and initial page can't be written one.
- **routes** - routes are used to create a dictionary type list with key and page. That means to navigate Screen0 we first have to specify in routes on main.dart

Navigator.pushNamed - This will navigate to the screen to where the content is pushed.

- Main.dart

```
import 'Screen0.dart'
void main() => runApp(MyApp());
Class MyApp extends StatelessWidget {
  @override
  Widget build (——) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (Context) => Screen0(),
        '/first': (Context) => Screen1();
      },
    );
  }
}
```

- Screen0.

```
Class Screen0 extend StatelessWidget {
  @ override
    RaisedButton(
      Color: colors.red,
      onPressed: () {
        Navigator.pushNamed(Context, '/first');
      },
    ),
```

→ In this program first names of diff. screens are declared on main.dart with the help of routes. Then initialRoute is used in place of home with routes to display initial screen number mentioned. Then onpressed will navigate to Screenfirst with the help of Navigator.

Async and sync functions.

Async - This is used to specify that the function is Asynchronous function

Await - It is used to tell the function that the function ~~be executed~~ should wait for the function to be completed.

Duration - Duration is used to set the time of display

Future Delayed - this function requires 2 parameters. First one duration and second one the task to be executed -

Future <String> - this will tell that the coming output will be of type String

```
Void main(){
    performTask();
}

Void performTask() async {
    task1();
    String task2Result = await task2();
    task3(task2Result);
}

Void task1(){
    String result = 'task 1 data';
    print (result);
}

Future<String> task2() async {
    Duration threeSeconds = Duration(seconds: 3);
    String result;

    await Future delayed (threeSeconds, () {
        result = 'task 2 data';
        print (result);
    });
    return result;
}

void task3 (task2Result) {
    print ('task 3 completed with $task2Result')
}
```

→ So, in this program performTask will execute the functions asynchronously. task1 will be executed then task2 function will be called and will wait for 3 seconds to execute the statement. This will not jump to next task until await function is over. After that 3 seconds 'task2data' and this statement will be executed.

Lifecycle methods

initstate - This method is used to ~~call the~~ be called running the app. everything inside it will get executed on run.

deactivate - This method is used to be called while the app is closed or when moving from that screen to another.

```
import 'package: flutter/material.dart';

class LoadingScreen extends StatefulWidget {
  @override
  _LoadingScreenState createState() => _LoadingScreenState();
}

class _LoadingScreenState extends State<_> {
  @override
  void initState() {
    super.initState();
    getLocation();
  }

  void getLocation() async {
    Position position = await GeoLocator()
      .getCurrentPosition(desiredAccuracy:
        LocationAccuracy.low);
    print(position);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold();
  }
}
```

So in this when app is made to run first of all init method is called an will request the location without doing any further task. If deactivate method is used then at run at first init and build is called then after closing app or switching the screen all the functionality in deactivate method will be executed.

Exception handling and null aware.

try - This will execute the statement inside it and will try to run it, if its correct it will move to next execution or otherwise send the error to catch.

Catch - Catch will get the error showed by try and will display in console without breaking code to work.

?? - This is used here and is checking if the variable before is null or occupied. if occupied it will assign the ~~no~~ int after it to that vari.

Throw - Throw will display the message assigned to it when error is thrown.

Parse - Parse is used to convert string to double or anything to anything

(i)

```
@override
Widget build(BuildContext context){
  String myMargin = 'abc';
  double myMarginAsADouble;

  try {
    myMarginAsADouble = double.parse(myMargin);
  } catch (e) {
    print(e);
  }

  return Scaffold(
    body: Container(
      margin: EdgeInsets.all(myMargin -
                                AsDouble ?? 30.0),
      Color: Colors.red,
    ),
  );
}
```

(ii)

```
void location() async {
  try {
    Somethingthatexpect(12);
  }
  catch (e) {
    print (e);
  }
}
void somethingthatexpect(int n){
  if(n>10){
    throw 'n is greater than 10.';
  }
}
```

→ In first program it will check if statement in try give error if yes it will be sent to catch and will displayed to ~~catch~~ console without breaking compiling. Throw will show message that you need to show and ?? will ensure that the value should no be null if yes then the number after it will be assigned to var.

Using API

as - It is used when there are multiple import statements and to remember from where the methods inside the files are called from.

http.Response - As above mentioned http is used here as we used as keyword to denote that file from which Response is called.

Response - It is a data type that stores the response got.

get - It is used to send Api request to server.

Uri parse - This requires an url to locate and request the info needed

response.body - this will extract body of info received.

```
import 'package:http/http.dart' as http;

Class LoadingScreen extends StatefulWidget {
  @override
  _LoadingScreenState CreateState() =>
                                _LoadingScreen
                                State();
}

Class _LoadingScreenstate extends State <_> {
@override
void initstate() {
  Super.initState();
  getLocation();
  getData();
}

Void getData() async {
  http.Response response = await http.get
  (uri.parse('http://api _____'));
  if (response.StatusCode ==200) {
    String data = response.body;
    print (data);
  } else {
    print (response.Statucode);
  }
}
```

Uniform resource [redacted] identifier

→ So in this code http package is used to recieve result from server. Response is a [redacted] Class that has uriparse as a method to get info from server. and then response.body and status code is used to extract the specific info from info recieved.